IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 22, 2017, accepted December 28, 2017, date of publication January 15, 2018, date of current version March 9, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2790381

Weighted Greedy Dual Size Frequency Based Caching Replacement Algorithm

TINGHUAI MA[1,2], (Member, IEEE), JINGJING QU[1], WENHAI SHEN[3], YUAN TIAN[4], ABDULLAH AL-DHELAAN[3], AND MZNAH AL-RODHAAN[4]

[1]School of Computer Software, Nanjing University of Information Science and Technology, Nanjing 210-044, China
[2]CICAEET, Jiangsu Engineering Centre of Network Monitoring, Nanjing University of Information Science and Technology, Nanjing 210-044, China
[3]National Meteorological Information Center, Beijing100-080, China
[4]Computer Science Department, College of Computer and Information Sciences, King Saud University, Riyadh 11362, Saudi Arabia

Corresponding author: Tinghuai Ma (thma@nuist.edu.cn)

This work was supported in part by the Special Public Sector Research Program of China under Grant GYHY201506080, in part by the National Science Foundation of China under Grant 61572259 and Grant U1736105, in part by PAPD, and in part by the Deanship of Scientific Research at King Saud University for funding this work through research group under Grant RGP-VPP-264.

ABSTRACT Caches are used to improve the performance of the internet, and to reduce the latency of data access time and the low speed of repeated computing processes. Cache replacement is one of the most important issues in a caching system; therefore, it must be coordinated with the caching system to minimize the access latency and maximize the hit rate or byte hit rate. In this paper, we presented a novel caching replacement algorithm named Weighted Greedy Dual Size Frequency (WGDSF) algorithm, which is an improvement on the Greedy Dual Size Frequency (GDSF) algorithm. The WGDSF algorithm mainly adds weighted frequency-based time and weighted document type to GDSF. By increasing the above two weighted parameters, WGDSF performs fairly well at keeping popular objects in the cache and replacing rarely used ones. Our experiment shows that this algorithm has a better hit rate, byte hit rate and access latency than state-of-the-art algorithms, such least Recently Used, least Frequently Used, and GDSF.

INDEX TERMS Cache replacement, hit rate, weighted frequency.

I. INTRODUCTION

When two different-speed devices need collaboration work, the high-speed device always waits for the low-speed device. The collaboration performance of two devices is decided by the low-speed device. As we all know, there is a huge latency gap between processors and memory; this gap causes a lot of inconvenience when using the processors, and the system cannot make full use of the computing power of processors. To resolve this problem, system designers used to add last level cache (LLC) to hide the latency gap. Many state-of-the-art cache replacement algorithms of LLC have been proposed [1]–[5]. In the network environment, the logic position of cache is generally located between the data source and the data requester. The cache technique utilizes a certain algorithm to copy some data from the data source to another storage medium temporarily close to the data requester [6]. The research on web caches mainly focuses on the proxy server cache [7], [8], the wireless network [9]–[15] and the content-centric networking [7], [16]–[21] at home and abroad. By deploying the cache into the architecture, the architecture efficiency is improved. Some papers address the performance evaluation of the architecture with various cache replacement algorithms [6], [22], [23].

Several factors have been used to decide on the data that is to reside in cache memory, including include cache coherence [24], admission and replacement policies [25]. Therefore, this technology can be used at various levels [26]. There are four classes classified by cache purpose (transaction cache, processes cache, query cache, and cluster cache), and they have different application scenarios. For example, in the national meteorological big data environment, the long distances of the physical communication links among meteorological data nodes results in a bit limit on the data query, which adds a large amount of network missing packages [27]. Furthermore, it makes the transmission status quite complex and makes it very difficult to achieve the requirement of real-time data sharing. To resolve this issue, we mainly use the query cache and propose a cache replacement algorithm in our paper, with the aim of reducing the access frequency of the physical data source. When the application software

accesses the data in the database table, it goes to cache first. If the data is found in cache, the accessing speed will be greatly improved. Otherwise, it goes to the data sources and copies the data into the cache in preparation for accessing it later on. In summary, the performance of application software will be improved through the assistance of query cache.

Existing replacement algorithms may not be ideal with respect to cache hit rate and byte hit rate. The algorithms are classified as recently based algorithms, frequency-based algorithms, size-based algorithms or utility value-based algorithms. The typical algorithms that are based on the value model are LRV [28] and Greedy Dual Size algorithms [29]. The variants of Greedy Dual Size cache replacement algorithms maximize the different performance metrics. GDSF [28] is well known for its prominent hit rate. Therefore, we made some improvements on the basis of GDSF. We studied all parameters that have an impact on cache performance and designed a priority queue for objects in cache. Based on the weighted frequency base time and weighted document type, we utilized the priority queue to keep the popular objects in cache and to improve the performance of the cache system. More discussion about related work on caches will be presented in Section 2. Section 3 focuses on describing the variants of Greedy Dual algorithms. In Section 4, we will describe the novel algorithm we have proposed. In Section 5, we will present the experimental setup for comparing our algorithm with the state-of-the-art algorithms, and carry out the experiment plan.

II. RELATED WORK

Jeong and Dubois [30] gave an exhaustive survey of cache replacement algorithms proposed for web caches through the year 2003. They put forward a simple classification scheme that can used to describe and evaluate the cache replacement algorithms. Furthermore, they listed which fields of caching algorithms are worthy of research. Podlipnig and Boszormenyi listed more than 50 cache policies and suggested policies for different environments [31]. Wong [32] mainly compared the performance of the recency algorithm, frequency algorithm and the recency/frequency algorithm running the same data sets [32]. Wong [32] gave a comparison of different cache replacement policies in traditional systems as well as in web applications and proposed a system that implements LRU and CERA caching algorithms. Their contribution was to try to apply the traditional cache replacement algorithms to the web. The C-Aware algorithm is an example of an algorithm applied to the web; it was proposed and devised by Khandekar and Mane*et al* [33]. As we all know, there is a multitude of cache replacement policies, which are suitable for different scenarios. We classify the existing cache replacement policies into four groups, which are recently based policies, frequency-based policies, size-based policies and utility value-based policies.

A. RECENTLY-BASED POLICIES

Recently-based policies decide whether or not a document is to be replaced based on whether it is a recently referenced object. They perform well when many users are interested in the same objects at about the same time. The available replacement policies are LRU [34], LH-MLRU [35], LLC [36], etc. Most of the time, the hit rate and byte hit rate of the recently-based policies are good. In addition, their complexity is fair.

LRU [34] is a representative policy of the recently-based policies, and it replaces the object that was requested least recently. This traditional policy is the most frequently used in practice and has worked well for CPU caches and virtual memory systems. However, it does not work as well for proxy caches because the locality of time accesses for Web traffic often exhibits very different patterns.

B. FREQUENCY-BASED POLICIES

Frequency-based policies decide whether or not a document is to be replaced based on whether it is a frequently referenced object. They perform well when many users tend to access objects with quite steady popularities. The available replacement policies are LFU [34], LFU-DA [30], PLFU [37], etc. The hit rate and byte hit rate of the frequency-based policies are acceptable, and their complexity is also not high.

LFU [35] is a representative policy of the frequency-based policies; it replaces the object which has been accessed the fewest times. This algorithm tries to keep more popular objects and replace rarely used ones. However, some documents can build a high frequency count and never be accessed again. The traditional LFU algorithm does not provide any mechanism for removing such documents, and this leads to cache pollution.

C. SIZE-BASED POLICIES

Size-based policies decide whether or not a document is to be replaced based on whether it is a small object. They perform well when many users tend to access information-based objects. The available replacement policies are SIZE [28], etc. The size-based policies' hit rate and byte hit rate are fair, and their complexity is adequate.

SIZE [28] is a representative policy of the size-based policies; this algorithm tries to minimize the miss ratio by replacing one large document rather than a bunch of small ones. However, some of the small documents brought to a cache may never be accessed again. The SIZE algorithm does not provide any mechanism for replacing such documents, which leads to pollution of the cache.

D. UTILITY VALUE-BASED POLICIES

Utility value-based policies decide whether or not a document is to be replaced based on how much value the object has. They perform well when the system has sufficient processing and memory resources. The available replacement policies are GD, GDS [29], LRV, GDSF [28], GDSF-AI [38], etc. The variants of Greedy Dual Size algorithms are well known for their abilities to maximize different performance metrics. Example metrics include hit ratio and byte hit ratio. In summary, the hit rate and byte hit rate of the utility value-based

TABLE 1. Existing cache replacement algorithms.

algorithm based	recently	frequency	size	function	random
replacement rationale	recently referenced document	frequently referenced document	smaller document	higher function value document	stochastic document
application scenarios	system which is accessed at the same time	system with quite steady popularity	information based system	system has sufficient performance	system has limited performance
available algorithms	LRU [35], LH-MLRU [36],LLC[37]	LFU [35], LFU-DA [31],PLFU [38]	SIZE [29]	GD [30], GDS [30], GDSF [29], GDSF-AI [39], LRV [29]	Harmonic [31]
representative algorithm	LRU[35]	LFU[35]	SIZE [29]	GD [30]	Harmonic [31]
hit rate	Fair	Fair	Fair	Best	Worst
byte hit rate	Fair	Fair	Fair	Best	Worst
complexity	Fair	Fair	Fair	Highest	Lowest

policies are the best, whereas their complexity is slightly higher.

We will introduce GD series algorithms of utility value-based policies in Section 3.

E. RANDOM-BASED POLICIES

Random-based policies decide whether or not a document is to be replaced based on whether it is a stochastic document. They perform well when the system has limited performance. The available replacement policies are Harmonic and Random [30]. The random-based policies have the worst hit rate and byte hit rate, but their complexity is the lowest. Harmonic [30] is a representative policy of the random-based policies; this algorithm tries to select replacement documents randomly from the cache. Therefore, its complexity is fair, and its performance is stable. In summary, Table 1 lists the replacement polices reported in the literature according to their categories. There are three main aspects to be considered. The first part is the replacement algorithm design rationale behind the categories and which category works particularly well in which scenarios. The second part is the available replacement algorithms and representative algorithm for each category. The last part is the overall performance of different categories in terms of different metrics.

III. GREEDY DUAL ALGORITHM

A. THE ORIGINAL GREEDY DUAL ALGORITHM

The original Greedy Dual algorithm was introduced by Young [30]. It addresses the case in which documents in a cache (memory) have the same size but have different costs to fetch them from the secondary storage. The algorithm associates a value, H, with each cached document p. Initially, when a document is brought to cache, H is defined as the standard cost of taking the document into the cache. When a replacement needs to be made, the document with the lowest H value, $\min_H$, is replaced, and then all the documents in the cache reduce their cost values H by $\min_H$. If a document p is accessed again, its current cost value H is restored to the original cost. In such a way, the H values of recently accessed documents maintain a larger amount of the original cost than the H values of documents that have not been accessed for a long time. The algorithm selects the documents with the lowest value $\min_H$ to be replaced first.

B. GREEDY DUAL SIZE ALGORITHM

The Greedy Dual Size algorithm (GDS)is a generalization of the Greedy Dual algorithm [30], which addresses uniform-size variable-cost objects. Using a priority queue and creative joined offset value L for future settings of H, Rizzo and Vicisano [38] proposed a new algorithm named Greedy Dual Size (GDS). The priority queue key for a document i is computed in the following way: $H(i) = L + \frac{Value(i)}{Size(i)}$.

The parameter L is a running queue that starts at 0 and is updated with the value of the lowest value $\min_H$ in the priority queue. Therefore, the GDS algorithm need not have all the documents in the cache reduce their cost values H by $\min_H$. The parameter $Size(i)$ is the size of document i. The parameter $Value(i)$ is the cost associated with bringing document i to a cache; the author sets $Value(i) = 2 + Size(i)/536$, which is the estimated number of network packets sent and received to satisfy a cache miss for a requested number, where 536 is the default maximum TCP segment size.

C. GREEDY DUAL SIZE FREQUENCY ALGORITHM

The Greedy Dual Size Frequency algorithm (GDSF) combines the GDS algorithm with access frequency. The GDSF algorithm performs well in most scenarios. GDSF also uses the priority queue. The priority queue key for a document is computed in the following way [29]:

$$H(i) = L + Fr(i) * \frac{Value(i)}{Size(i)} \tag{1}$$

The parameters L, $Value(i)$and $Size(i)$ are the same as those in the GDS algorithm. The parameter $Fr(i)$ is a document i frequency count.

D. GREEDY DUAL SIZE FREQUENCY ACCESS INTEREST ALGORITHM

The GDSF-AI [39] algorithm is based on the GDSF algorithm and combines web users' interest, content types

and the access characteristics of the target of the web. Web usersínterest is an important factor for effective web cache replacement policies. The array $k_{i1}, k_{i2}, \cdots, k_{in}$ is the keywords which represents *ith* web user's interest. The array $wq_{i1}, wq_{i2}, \cdots, wq_{in}$ is the list of weights corresponding to $k_{i1}, k_{i2}, \cdots, k_{in}$. The GDS-AI algorithm represent the *ith* web user' interest model as the vector $q_i = \{< k_{i1}, wq_{i1} >, < k_{i2}, wq_{i2} >, \cdots, < k_{in}, wq_{in} >\}$. Instead of considering the usersąŕ interest keywordsąŕ distinctness, they consider all web users to have the same keywords; however, web users assign different keywords different weights.

The documents in cache can be represented as the vector $D = \{d_1, d_2, \cdots, d_m\}$. In this vector, d_j can be represented as:

$$d_j = \left\{< k_1, wd_{j1} >, < k_2, wd_{j2} >, \cdots, < k_n, wd_{jn} >\right\} \quad (2)$$

The array $k_1, k_2, \cdots, k_n$ is the list of keywords that includes all web users' interest. The array wd_{j1}, $wd_{j2}, \cdots, wd_{jn}$ is the list of the keywords' weights in the *jth* document corresponding to one user. The number wd_{j1} is the weight of keyword k_1 which is determined by the number of times it appears in the *jth* document. Then, given the interest vector q_i of web user i and the vector d_j of the *jth* document, they obtain the two term approximation degree of the documents by using the cosine formula as follows:

$$SIM(q_i, d_j) = \frac{\sum_{k=1}^{n} (wd_k * wq_k)}{\sqrt{\sum_{k=1}^{n} \left(wd_k{}^2\right)} * \sqrt{\sum_{k=1}^{n} \left(wq_k{}^2\right)}} \quad (3)$$

The function which represents the users interest has been presented in the following way:

$$Score(q_i, d_j, fr(j)) = SIM(q_i, d_j) * e^{\alpha(fr(i)-1)} \quad (4)$$

The priority queue key for a document is computed as:

$$H(j) = L + V_{type} * \left(\frac{P(j) * Value(j)}{\log(Size(j))}\right) * Score(q_i, d_j, fr(j)) \quad (5)$$

The parameter L, $Value\,(j)$, $Size\,(j)$ and $fr(j)$ are same as GDSF algorithm. The parameter $P(j)$ is the probability that the document will be accessed again.

IV. PROPOSED ALGORITHM

It is impossible to hold all documents in the cache. When a new document is to be put into the cache, a rule needs to be applied to replace some documents. The design of replace rules is what we need to do. In this paper, we proposes a new replacement algorithm named Weighted Greedy-Dual-Size-Frequency (WGDSF), which is based on the GDSF algorithm and joined the weighted frequency based time parameter and weighted document type. Compare our algorithm with the state-of-the-art algorithms, this algorithm has not only higher document Hit Rate and Byte Hit Rate, but also lower access latency than GDSF.

A. WEIGHTED FREQUENCY BASED TIME

Weighted frequency based time parameter is the key word which represents content popularity, it also counts for much in the process of cache replacement. As the longer the document is not used after the last visit, the probability of accessing the document again is smaller. That is to say, the importance of the document falls. What we want to do is finding out these documents and removing these documents to the storage place of low performance. The parameter of time we used is T_i. The implication for T_i is as follows: we can define a time array $t_1, t_2, \ldots, t_n$ of the each visit or change time, starting from the document generation. After that, we can calculate the distances between these time points t_i and the next time t_{i+1}, by the computational formula $t_2 - t_1, t_3 - t_2, \ldots, t_{i+1} - t_i, \ldots, t - t_n$. The parameter t is the current time. According to the value $T_i = t_{i+1} - t_i$, we can get corresponding time interval T_i. For one document j, its first time stamp in cache is t_1, then, while time stamp is t_2, we know $T_i = t_2 - t_1$. If we consider time stamp as t_1, while document j is the same time locate in cache, so there is no hit count, it need not to calculate T_i. During the time interval, a series of cache hit behaviors may occur, and the corresponding hit counts named F_i. Now, we can give the weighted frequency function of the *jth* document in the following way:

$$WTF(j) = \sum_{i=1}^{fr(j)} \frac{F_i}{T_i} \quad (6)$$

The reason for this is to distinguish the hit counts happened nearly a period of time or the hit counts not. For example, two documents have the same hit count number during a period of time, but one is less recently used and the other one is more recently used. The weighted frequency based on time parameter can use a weighted formula to represent the difference.

For each time period, we divide the document access frequency in *ith* time period F_i by the interval of time T_i. Then, we can get $WTF(j)$ by adding all the periods' value together.

B. WEIGHTED DOCUMENT TYPE

Meteorological data has some distinguishing features, like complicated sources, various and diverse formats, different forms, and substantial data. Due to the rare-updated feature that meteorological data has, the parameter weighted document type (WDT) is also rare-updated which represents the proportion of meteorological data varieties. Documents have a particularly large number of varieties, such as meteorological texts, meteorological images, and meteorological videos, meteorological audios. Among the meteorological texts, the documents take a greater proportion whose data variety is nc and the grib.

We record each access time, and design a caching replacement algorithm, which can keep the weightiest data in the cache system. We can give the function which represents the document's weighted type in the following

way:

$$WDT(j) = \frac{Count(judge(j))}{Count_{total}} \tag{7}$$

For *jth* document, we judge the document belongs to what type according to the function *judge*(*j*). The *judge*(*j*) means we classify the document j into text, image, videos, radar data etc. types. We calculate the number of documents of this type through $Count(judge(j))$. Dividing $Count(judge(j))$ by the whole number of documents , we can get a weight value WDT of this type documents $Count_{total}$. If the WDT of document type is big, it shows that this type of documents is more important and this document is more important than other types of documents.

C. SIZE AND COST

We can give the function which represents the document's size and cost value in the following way:

$$SC(j) = \frac{Cost(j)}{\log(Size(j))} \tag{8}$$

The parameter $Size(j)$ is the size of *jth* document. As the meteorological data is vary from text to image. So the size(j) can be varied from Kb to Gb. The $SC(j) = Cost(j)/Size(j)$ will result in a big domain field. To avoid the size numerical value's serious uneven distribution, we use log transformation to make it in accordance with our assumption, namely we take $log(Size(j))$ as document size. The parameter $Cost(j)$ is the cost associated with document to cache it. Here, we set cost function for each document to $Cost(j) = 2+Size(j)/536$. The parameter $Cost(j)$ is the estimated number of network packets sent and received to satisfy a cache miss for requested number. Furthermore, 536 is the default maximum TCP segment size.

D. PROCESS AND ALGORITHM

As described above, the high cache hit ratio of the data objects with long access time from devices result to high performance gain. We define a document value calculating function for each object in cache as the following:

$$H(j) = L + SC(j) * WDT(j) * WTF(j) \tag{9}$$

The parameter L is the expansion factor as same as the L of GDSF algorithm. So WGDSF need not all the documents in the cache to reduce their cost values H. The parameter $SC(j)$, $WDT(j)$ and $WTF(j)$ are as above described. The caching process of WGDSF algorithm is illustrated in Fig.1.

When the system receives the request for j document, the WGDSF algorithm will check whether the document j exists in the cache. If the document is in the cache, cache responses the document j to the request. If the document is not in cache, WGDSF requests the original data source firstly and responses the document j to the request. After that, WGDSF calculates the value $H(j)$ of the j document by using the function $H(j) = L + SC(j) * WDT(j) * WTF(j)$ check whether the remaining cache has enough space to place the

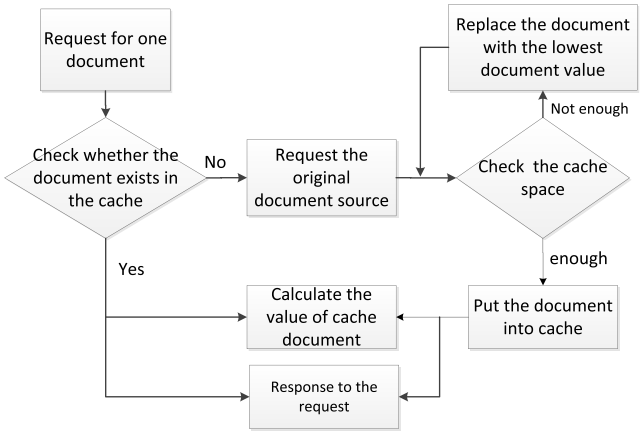


FIGURE 1. The caching process.

document j. Then, the used space $Used$ plus $Size(j)$ equals the updated used space. If the used space $Used$ is less than cache size, WGDSF puts the document into cache.

Next, we introduce the Insufficient space condition, which is as the same meaning as formula $Used \leq Total$. We recalculate the value of all cached documents and choose the documents $\{m_1, m_2, \cdots, m_k\}$, which have lowest $\min_H$. The reason we recalculate all cached documents' H value is that weighted frequency and weighted document type need to be updated. The documents satisfy the conditions that $Used - \sum_{i=1}^{k} Size(m_i) \leq Total$ and $H(m_1) \leq H(m_2) \leq \ldots \leq H(m_k)$. The reason we choose a series of documents is that evicting single document may not meet the cache space needs of document j. It must be stressed that document j should be included in $\{m_1, m_2, \cdots, m_k\}$ if document j has the lowest $\min_H$.

Next, we check whether the document j exists in $\{m_1, m_2, \cdots, m_k\}$. If document j is in $\{m_1, m_2, \cdots, m_k\}$, that is to say, the value of document is low and it is not necessary to put document in the cache. Therefore, WGDSF discards the document j directly. This step promises not to put low value document in the cache. If document j is not in $\{m_1, m_2, \cdots, m_k\}$, WGDSF discards the document set $\{m_1, m_2, \cdots, m_k\}$and fetch document j physically. Last but not least, WGDSF updates H for the lowest value $H(m_k)$, which is the highest value $H(m_k)$ in $\{m_1, m_2, \cdots, m_k\}$. This step promises not to put documents whose value lower than $H(m_k)$ in the cache and keep high value documents in cache.

More replacement detail is introduced as following Algorithm 1. The meanings of involving the parameters are as the following Table 2.

V. EXPERIMENT

A. DATASET AND EXPERIMENT ENVIRONMENT

We collected 6000 records (approximately 500 M) of meteorological data from the China Meteorological Data Sharing Service System (CMDSSS) [41]. Meteorological documents have many kinds of types, such as meteorological texts,

Algorithm 1 The WGDSF Algorithm Process

Input: the request for j document;
Output: j document;
Parameter: $L = 0$, $Used = 0$;

1: **if** j is in cache **then**
2: $fr(j) = fr(j) + 1$;
3: $H(j) = L + SC(j) * WDT(j) * WTF(j)$;
4: return j.
5: **else**
6: Obtain j from data source;
7: $fr(j) = 1$;
8: $H(j) = L + SC(j) * WDT(j) * WTF(j)$;
9: $Used = Used + Size(j)$;
10: **if** $Used < Total$ **then**
11: Fetch j.
12: **else**
13: We recalculate the H value of all cached documents and choose documents $\{m_1, m_2, \cdots, m_k\}$, which have lowest value, H,and they meet the conditions below:
14: $Used - \sum_{i=1}^{k} Size(m_i) \leq Total$;
15: $H(m_1) \leq H(m_2) \leq \ldots \leq H(m_k)$;
16: **if** j is in **then**
17: Discard j.
18: **else**
19: $L = \min_H = \max_{i=1}^{k} H(m_i) = H(m_k)$
20: $Used = Used - \sum_{i=1}^{k} Size(m_i)$
21: Evict the minimum $\{m_1, m_2, \cdots, m_k\}$,and fetch j.
22: **end if**
23: **end if**
24: **end if**

TABLE 2. The meanings of involving the parameters.

L	the expansion factor
$Used$	the cache space already used
$Total$	the total space of cache store
$fr(j)$	the frequency of document j
$H(j)$	the value of document j
$Size(j)$	the size of document j
$SC(j)$	the document j overall effect value of Size and Cost
$WDT(j)$	the weighted document type of document j
$WTF(j)$	the weighted frequency based time of document j

meteorological images, and meteorological videos, meteorological audios. Among the meteorological documents, the text files take a greater proportion whose size is about several kb. This huge small size text documents are very important for weather prediction, and be visited frequently and also occupy small space. So, we cache the huge small text files is effective for data transfer. We consider to cache the while documents rather than documents indexes. The experimental environment is based on a Lenovo server, 4G RAM, running Windows 7. Based on the open source java project EHCache [40], which integrated the LRU, LFU and FIFO algorithms, we added the GDSF algorithm and WGDSF algorithm into the Ehcache project to make contrast experiment. The GDSFAI algorithm needs a user interest database and its scenario is a TV system, so we did not add GDSFAI to Ehcache and the contrast experiment.

B. PERFORMANCE METRICS

We compare the performance of the algorithms LRU, LFU, GDSF and WGDSF based on cache hit rate, byte cache hit rate and average access latency.

1) CACHE HIT RATE

Cache hit rate is computed in the following way:

$$HR = \frac{\sum_{i=1}^{N} q_i}{N} \tag{10}$$

Hit rate (HR) is a widely used performance evaluation criterion. The parameter q_i is a Boolean value of the document: a zero value indicates a present miss, and a nonzero value indicates a present hit. Then we can obtain the cumulative hit count by adding all q_i values. Now we can compute the cache hit rate by dividing the cumulative hit count by the total count of the request. The parameter N is the total hit count.

2) CACHE BYTE HIT RATE

Cache byte hit rate is computed in the following way:

$$BHR = \frac{\sum_{i=1}^{N} q_i s_i}{\sum_{i=1}^{N} s_i} \tag{11}$$

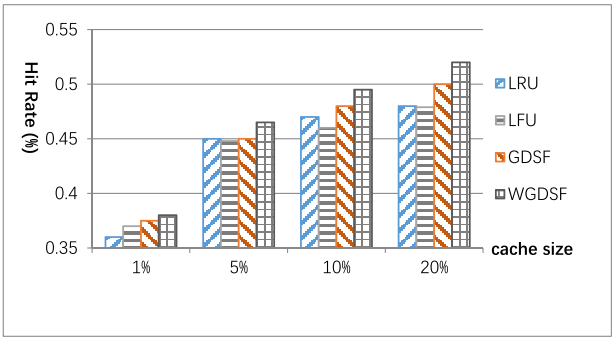
Byte hit rate (BHR) is the other widely used performance evaluation criterion. The parameter q_i is the same as described in the discussion of Hit Rate. The parameter s_i is the size of *ith* document. Then, we can obtain the cumulative byte hit value by adding all values of q_i multiplied by s_i. We can also obtain the total value of the request by adding all object sizes. Now we can compute the cache byte hit rate by dividing the cumulative byte hit value by the total value of the request.

3) AVERAGE ACCESS LATENCY

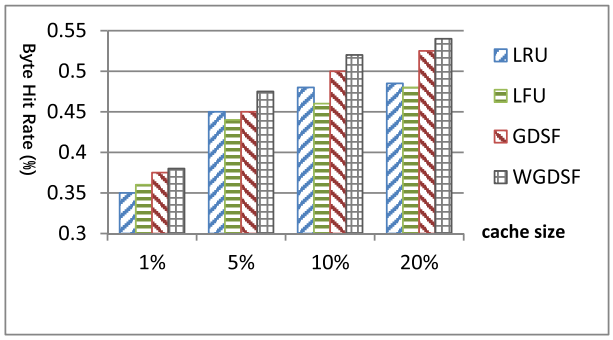
Average Access Latency (AAL) is computed in the following way:

$$Latency = \frac{\sum_{i=1}^{N} T_i}{N} \tag{12}$$

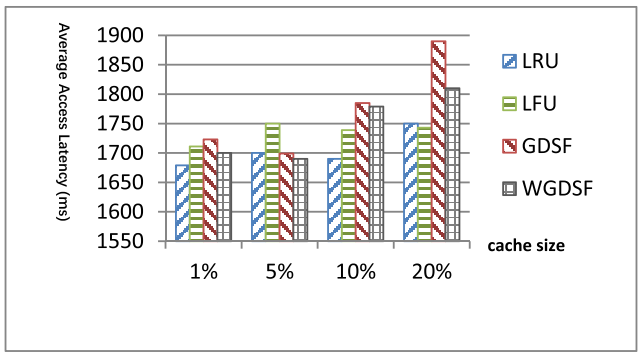
Average access latency can represent the cache systemąŕs capacity for reducing the access latency, which is computed by dividing the general request time by the request frequency. The parameter T_i is the time interval of *ith* request. Because low latency is the goal of adding a cache replacement algorithm, AAL is another import performance metric except for that of HR and BHR. The lower AAL value illustrates that the cache replacement algorithm plays a better role.



(a)

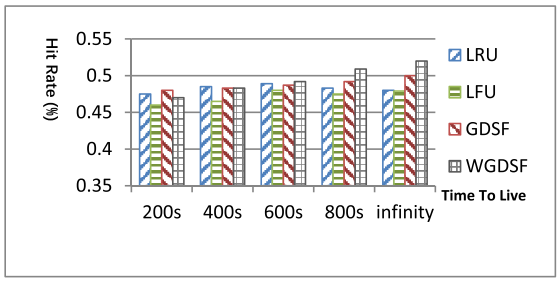


(b)

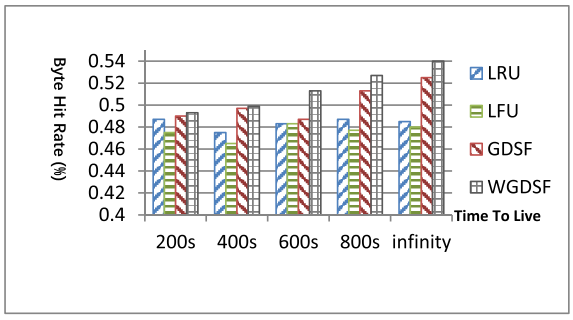


(c)

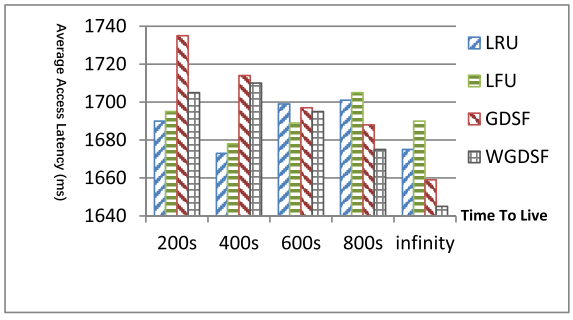
FIGURE 2. (a) Algorithm HR performance of different cache size; (b) Algorithm BHR performance of different cache size; (c) Algorithm AAL performance of different cache size.



(a)



(b)



(c)

FIGURE 3. (a) Algorithm HR performance of different Time To Live; (b) Algorithm BHR performance of different Time To Live;(c) Algorithm AAL performance of different Time To Live.

C. RESULT AND ANALYSIS

We choose three traditional cache replacement algorithms (LRU, LFU and GDSF) to compare with the policy we proposed. What we will do is compare the respective performances of different algorithms with respect to different cache size, time to live and user access pattern.

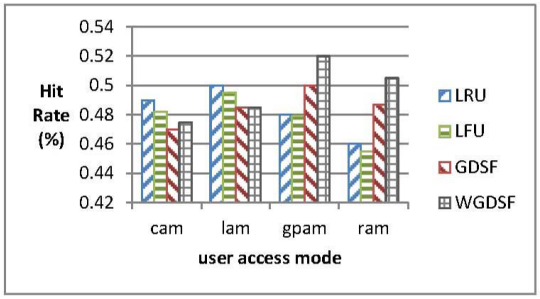
1) IMPACT OF CACHE SIZE

In this experiment set, we compare the performance results of different algorithms across a wide range of cache sizes (cache capacity percentage of data set size), from 1 percent to 20 percent. Furthermore, the parameter time to live (TTL) and the access pattern remain unchanged. The parameter TTL has a default value of 0, which means that cache document can live indefinitely. The user access pattern keeps global probabilistic access mode (GPAM). All of this can be adjusted by editing the configuration files, which are .xml files.

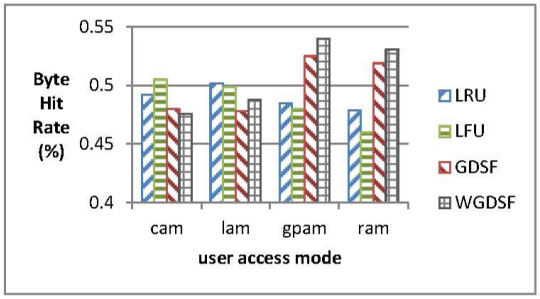
Figure 2 shows that WGDSF has higher percentages of HR and BHR than those of GDSF and other traditional algorithms. For the performance metric AAL, we see in Figure 2(c) that WGDSF had the best possible performance for the cache size of 5 percent. Further, we can reach the preliminary conclusion that WGDSF performs better when the cache size is smaller. Although, sometimes the AAL is slightly higher than for other algorithms, WGDSF is approximately 3 percent higher than conditions with no cache.

2) IMPACT OF CACHE TTL

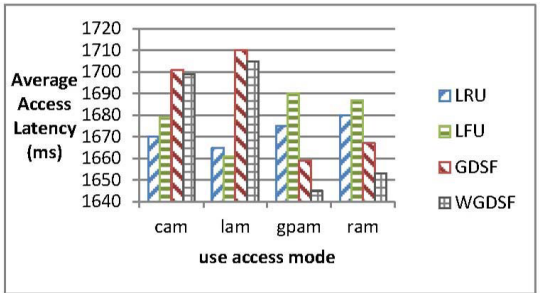
In this experiment set, we compare the performance results of different algorithms across a wide range of time to live (TTL), from 200s to 1200s. Furthermore, the parameter cache size



(a)



(b)



(c)

FIGURE 4. **(a) Algorithm HR performance of different user access pattern; (b) Algorithm BHR performance of different user access pattern; (c) Algorithm AAL performance of different user access pattern.**

and access pattern remain unchanged. The cache size parameter remains at 5 percent. The user access pattern remains in global probabilistic access mode. All of this can be adjusted by editing the configuration files, which are .xml files. Figure 3(b) shows that WGDSF has fairly good BHR performance gain over GDSF for a wide range of TTL values. We can see in Figure 3(a) that for TTL values bigger than 600, WGDSF has a higher percentage of HR than other traditional algorithms. Because a short TTL time will affect the document value definition of WGDSF and GDSF, we see in Figure 3 that WGDSF has a poor performance when the TTL value is smaller than 600. Although, sometimes the AAL is slightly higher than for other algorithms, WGDSF is just about three percent higher than conditions with no cache.

The experimental results and application background explain why WGDSF has poor performance. The high value documents which WGDSF predicts will be removed from the cache system if the TTL is too short, which seriously affects the performance of WGDSF and results in the document value descending and the document being replaced if the document live time increases to more than 600s. We consider that the condition described above can explain the poor performance.

3) IMPACT OF USER ACCESS PATTERN

In this experiment set, we compare the performance results of different algorithms across a wide range of user access patterns, which includes continuous access mode (cam), loop access mode (lam), global probabilistic access mode (gpam) and regional access mode (ram). Furthermore, the parameters of cache size and TTL remain unchanged. The cache size parameter remains at 5 percent. The parameter TTL is set to default 0, which means that cache document can live for infinity. All of this can be adjusted by editing the configuration files, which are .xml files.

Figure 4 shows that WGDSF has fairly good HR, BHR and AAL performance gain over GDSF for global probabilistic access mode (GPAM) and regional access mode (RAM). In continuous access mode (CAM) and loop access mode (LAM), LRU and LFU have a higher performance than GDSF and WGDSF. However, in real networks, global probabilistic access mode is the most common, and regional access mode is the second most common. Therefore, WGDSF can be put into use in real networks and can perform well.

VI. CONCLUSIONS

Weighted frequency-based time, weighted document type, document size and replacement cost are important factors for effective cache document replacement policies. This article proposes a new replacement algorithm named Weighted Greedy-Dual-Size-Frequency (WGDSF), which is based on the GDSF algorithm and joined with weighted frequency-based time and weighted document type. The experiment shows proposed algorithm especially performs better while the cache size is smaller. For documents' time to live, our algorithm has a higher percentage of HR than other traditional algorithms while TTL more than 600 seconds, which is suitable for caching. Future more, the proposed algorithm performances well in global probabilistic access mode, which is the most common in real network. So, WGDSF can be put into use in real networks and can get well performance.

REFERENCES

[1] P. Abad, P. Prieto, V. Puente, and J.-A. Gregorio, "Improving last level shared cache performance through mobile insertion policies (MIP)," *Parallel Comput.*, vol. 49, pp. 13–27, Nov. 2015.

[2] V. V. Fedorov *et al.*, "ARI: Adaptive LLC-memory traffic management," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 4, p. 46, 2013.

[3] C. T. Do *et al.*, "A new cache replacement algorithm for last-level caches by exploiting tag-distance correlation of cache lines," *Microprocess. Microsyst.*, vol. 39, no. 4, pp. 286–295, 2015.

[4] T. S. Warrier, B. Anupama, and M. Mutyam, "An application-aware cache replacement policy for last-level caches," in *Architecture of Computing Systems CARCS*. Prague, Czech Republic: Springer, 2013, pp. 207–219.

[5] N. Gast and B. van Houdt, "Transient and steady-state regime of a family of list-based cache replacement algorithms," *Queueing Syst.*, vol. 83, nos. 3–4, pp. 293–328, 2016.

[6] M. Gallo, B. Kauffmann, L. Muscariello, A. Simonian, and C. Tanguy, "Performance evaluation of the random replacement policy for networks of caches," *Perform. Eval.*, vol. 72, pp. 16–36, Feb. 2014.
[7] H. Elaarag, "Web proxy cache replacement scheme based on backpropagation neural network," *J. Syst. Softw.*, vol. 81, no. 9, pp. 1539–1558, 2008.
[8] W. Ali, S. M. Shamsuddin, and A. S. Ismail, "Intelligent Web proxy caching approaches based on machine learning techniques," *Decision Support Syst.*, vol. 53, no. 3, pp. 565–579, 2012.
[9] L. Yaom J. Deng, J. Wang, and G. Wu, "CACHE: An anchor-based public key caching scheme in large wireless networks," *Comput. Netw.*, vol. 87, pp. 78–88, Jul. 2015.
[10] Y. Lv *et al.*, "An efficient and scalable density-based clustering algorithm for datasets with complex structures," *Neurocomputing*, vol. 171, pp. 9–22, Jan. 2016.
[11] H. Chen, Y. Xiao, and S. V. Vrbsky, "An update-based step-wise optimal cache replacement for wireless data access," *Comput. Netw.*, vol. 57, no. 1, pp. 197–212, 2013.
[12] A. Karami and M. Guerrero-Zapata, "An ANFIS-based cache replacement method for mitigating cache pollution attacks in named data networking," *Comput. Netw.*, vol. 80, pp. 51–65, Apr. 2015.
[13] S. Vobugari, D. Somayajulu, and B. Subaraya, "Dynamic replication algorithm for data replication to improve system availability: A performance engineering approach," *IETE J. Res.*, vol. 61, no. 2, pp. 132–141, 2015.
[14] T. H. Ma *et al.*, "Detect structural-connected communities based on BSCHEF in C-DBLP," *Concurrency Comput., Practice Experim.*, vol. 28, no. 2, pp. 311–330, 2016.
[15] M. Akon *et al.*, "OUR: Optimal update-based replacement policy for cache in wireless data access networks with optimal effective hits and bandwidth requirements," *Wireless Commun. Mobile Comput.*, vol. 13, no. 15, pp. 1337–1352, 2013.
[16] I. Abdullahi, S. Arif, and S. Hassan, "Survey on caching approaches in information centric networking," *J. Netw. Comput. Appl.*, vol. 56, pp. 48–59, Oct. 2015.
[17] G. Gür, "Energy-aware cache management at the wireless network edge for information-centric operation," *J. Netw. Comput. Appl.*, vol. 57, pp. 33–42, Nov. 2015.
[18] S. Saha, A. Lukyanenko, and A. Ylä-Jääski, "Efficient cache availability management in information-centric networks," *Comput. Netw.*, vol. 84, pp. 32–45, Jun. 2015.
[19] T. H. Ma *et al.*, "LED: A fast overlapping communities detection algorithm based on structural clustering," *Neurocomputing*, vol. 207, pp. 488–500, Sep. 2016.
[20] T. H. Ma *et al.*, "KDVEM: A k-degree anonymity with vertex and edge modification algorithm," *Computing*, vol. 70, no. 6, pp. 1336–1344, Sep. 2015.
[21] H. Rong, T. H. Ma, M. L. Tang, and J. Cao, "A novel subgraph K+-Isomorphism method in social network based on graph similarity detection," *Soft Comput.*, to be published. [Online]. Available: https://doi.org/10.1007/s00500-017-2513-y
[22] F. Chao *et al.*, "Fast convergence caching replacement algorithm based on dynamic classification for content-centric networks," *J. China Univ. Posts Telecommun.*, vol. 20, no. 5, pp. 45–50, 2013.
[23] N. C. Fofack *et al.*, "Performance evaluation of hierarchical ttl-based cache networks," *Comput. Netw.*, vol. 65, pp. 212–231, Jun. 2014.
[24] T. Ma, C. Wu, W. Tian, and W. Shen, "The performance improvements of highly-concurrent grid-based server," *Simul. Model. Pract. Theory*, vol. 42, pp. 129–146, Mar. 2014.
[25] E. H. Cruz *et al.*, "Dynamic thread mapping of shared memory applications by exploiting cache coherence protocols," *J. Parallel Distrib. Comput.*, vol. 74, no. 3, pp. 2215–2228, 2014.
[26] J. Chen, V. Lee, and K. Liu, "Efficient cache management for network-coding-assisted data broadcast," *IEEE Trans. Veh. Technol.*, vol. 66, no. 4, pp. 3361–3375, Apr. 2017.
[27] B. Wu and A. D. Kshemkalyani, "Objective-optimal algorithms for long-term Web prefetching," *IEEE Trans. Comput.*, vol. 55, no. 1, pp. 2–17, Jan. 2006.
[28] T. H. Ma, W. Tian, and B. Wang, "Weather data sharing system: An agent-based distributed data management," *IET softw.*, vol. 5, no. 1, pp. 21–31, 2011.
[29] K. Li and H. Shen, "An improved GreedyDual cache document replacement algorithm," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell.*, Sep. 2005, pp. 457–460.
[30] J. Jeong and M. Dubois, "Cache replacement algorithms with nonuniform miss costs," *IEEE Trans. Comput.*, vol. 55, no. 4, pp. 353–365, Apr. 2006.
[31] S. Podlipnig and L. Boszormenyi, "A survey of Web cache replacement strategies," *ACM Comput. Surveys (CSUR)*, vol. 35, no. 4, pp. 374–398, 2003.
[32] K.-Y. Wong, "Web cache replacement policies: A pragmatic approach," *IEEE Netw.*, vol. 20, no. 1, pp. 28–34, Jan. 2006.
[33] A. A. Khandekar and S. B. Mane, "Analyzing different cache replacement policies on cloud," in *Proc. Int. Conf. Ind. Instrum. Control*, 2015, pp. 709–712.
[34] Z. Xudong *et al.*, "C-Aware: A cache management algorithm considering cache media access characteristic in cloud computing," *Math. Problems Eng.*, vol. 6, pp. 1–13, Sep. 2013.
[35] D. Lee *et al.*, "On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies," in *Proc. ACM Sigmetrics Int. Conf. Meas. Modeling Comput. Syst.*, 1999, pp. 134–143.
[36] J.-Q. Niu *et al.*, "Limited history based multi-LRU Web cache replacement algorithm," *J. Chin. Comput. Syst.*, vol. 6, p. 6, Mar. 2008.
[37] J. Reineke and D. Grund, "Sensitivity of cache replacement policies," *ACM*, vol. 12, no. 1s, pp. 1–18, 2013.
[38] L. Rizzo and L. Vicisano, "Replacement policies for a proxy cache," *IEEE/ACM Trans. Netw.*, vol. 8, no. 2, pp. 158–170, Apr. 2000.
[39] A. Sarhan, A. M. Elmogy, and S. M. Ali, "New Web cache replacement approaches based on internal requests factor," in *Proc. 9th Int. Conf. Comput. Eng. Syst. (ICCES)*, Cairo, Egypt, Dec. 2014, pp. 383–389.
[40] (Jul. 25, 2015). *EHCache*. [Online]. Available: http://ehcache.org/
[41] (May 25, 2015). *CMDSSS*. [Online]. Available: http://cdc.nmic.cn/home.do

TINGHUAI MA received the bachelor's and master's degrees from the Huazhong University of Science and Technology, China, in 1997 and 2000, respectively, and the Ph.D. degree from the Chinese Academy of Science, in 2003. He was a Post-Doctoral Associate with AJOU University, in 2004. From 2007 to 2008, he visited Chinese Meteorology Administration. In 2009, he was a Visiting Professor with the Ubiquitous Computing Laboratory, Kyung Hee University. He is currently a Professor in Computer Sciences with the Nanjing University of Information Science and Technology, China. He has authored over 100 journal/conference papers. His research interests are data mining, cloud computing, ubiquitous computing, and privacy preserving, and so on.

JINGJING QU received the bachelor's degree in computer science and engineering from the Nanjing University of Information Science and Technology, China, in 2012, where she is currently pursuing the Ph.D. degree. Her research interests include Grid Computing, Quantum Computing, and so on.

WENHAI SHEN received the bachelor's degree in mathematics from Fudan University, China, in 1982. He was with the National Climate Center, Chinese Academy of Meteorological Science, and the National Weather Center for over 30 years. He is currently a Professor with the National Meteorological Information Center. His research focus on meteorological information design and application.

YUAN TIAN received the master's and Ph.D. degrees from KyungHee University. She is currently an Assistant Professor with the College of Computer and Information Sciences, King Saud University, Saudi Arabia. Her research interests are broadly divided into privacy and security, which are related to cloud computing, bioinformatics, multimedia, cryptograph, smart environment, and big data. She is currently a member of technical committees of several international conferences. In addition, she is an active reviewer of many international journals.

ABDULLAH AL-DHELAAN received the B.S. degree (Hons.) in statistics from King Saud University, in 1982, and the M.S. and Ph.D. degrees in computer science from Oregon State University, in 1986 and 1989, respectively. He is currently the Vice Dean for Academic Affairs, the Deanship of Graduate Studies, and a Professor of Computer Science, King Saud University, Riyadh, Saudi Arabia. His current research interest includes: Mobile Ad Hoc Networks, Sensor Networks, Cognitive Networks, Network Security, Image Processing, and High Performance Computing. He is currently on the Editorial Boards of several journals and the organizing committees for several reputable international conferences. He has guest edited several special issues for the *Telecommunication Journal* (Springer) and the *International Journal for Computers and their applications*.

MZNAH AL-RODHAAN received the B.S. degree (Hons.) in computer applications and the M.S. degree in computer science from King Saud University, in 1999 and 2003, respectively, and the Ph.D. degree in computer science from the University of Glasgow, Scotland, U.K., in 2009. She is currently the Vice Chair with the Computer Science Department, College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia. Her current research interest includes: Mobile Ad Hoc Networks, Wireless Sensor Networks, Multimedia Sensor networks, Cognitive Networks, and Network Security. She has served in the Editorial Boards for some journals such as the *Ad Hoc journal* (Elsevier) and has participated in several international conferences.